Umelá inteligencia

Zadanie č.3 - Klastrovanie

(Zadanie 3b)

Autor: Marek Čederle
AIS ID: 121193 (xcederlem)
Cvičiaci: Ing. Nina Masaryková
(Streda 15:00)

Úvod

Ako spustiť program

Otvoríme si príkazový riadok v adresári so súborom a spustíme pomocou príkazu:
> python 3b.py

Po spustení program vypíše tieto inštrukcie pre pokračovanie:

```
Enter seed (random number 0-100): 40
Enter k (recommended <=20): 30
Enter max iterations (recommended 100 for centroid, 1 for medoid): 200
Enter choice (1 - k-means centroid, 2 - k-means medoid, 3 - divisive clustering centroid): 1
Clustering with k-means centroid:
Time taken: 5.875091075897217 seconds
Evaluate clustering:
Clustering successful
```

Opis problému:

Máme vygenerovať 20+40000 bodov, ktoré následne máme zhlukovať do k zhlukov podľa troch rôznych algoritmov:

- k-means, kde stred je centroid
- k-means, kde stred je medoid
- divízne zhlukovanie, kde stred je centroid

Nakoniec máme vyhodnotiť úspešnosť zhlukovača a vizualizovať výsledky.

Implementácia

Na implementáciu som použil jazyk Python a nasledovné programy:

- Vývojové prostredie (IDE) – PyCharm
- Dokumentácia – Microsoft Word

Použité knižnice:

- Numpy – na prácu s objektami bodov a ich poliami
- Matplotlib – na vykreslenie výsledkov

Opis riešenia

Poznámka:

- funkcie get_cost, manhattan_distance, k_means_medoid sú prevzaté zo zdroja č.7 a pozmenené pre moju implementáciu
- funkcia k_means_centroid je prevzatá zo zdroja č.2 a upravená podľa mojich pre moju implementáciu

generate_points(seed) – Vygeneruje požadovaný počet bodov

euclidean_distance(p1, p2) – Vypočíta Euklidovskú vzdialenosť dvoch bodov

manhattan_distance(p1, p2) – Vypočíta Manhattanovskú vzdialenosť dvoch bodov

get_cost(points, medoids) – Vráti clustre s bodami a cenu pre daný cluster

average_distance_from_point_to_points(points, centroid) – Vráti priemernú vzdialenosť bodov od centroidu

getPointsInCluster(points, labelx, cluster_id) – Vráti body, ktoré patria danému centroidu

evaluate_clustering(choice, points, oids, labels_here, k) – Vyhodnotí či clusterovanie bolo úspešné

k_means_centroid(points, k, max_iter)

```python
def k_means_centroid(points, k, max_iter):
    global labels
    # Initialize random k centroids
    centroids = np.random.uniform(np.amin(points, axis=0), np.amax(points, axis=0), size=(k, points.shape[1]))

    # Repeat until convergence
    for _ in range(max_iter):

        # Find the closest centroid
        labels = []
        for point in points:
            # calculate the distance between the point and each centroid
            distances = euclidean_distance(point, centroids)
            # assign the point to the closest centroid
            label = np.argmin(distances)
            # add the label for the point to the list of labels
            labels.append(label)

        # convert the list of labels to a numpy array
        labels = np.array(labels)

        # reposition centroids
        cluster_points = []
        for i in range(k):
            # find the points that belong to the cluster
            cluster_points.append(np.argwhere(labels == i))

        # calculate the new centroids
        cluster_centers = []
        for i, indices in enumerate(cluster_points):
            if len(indices) == 0:
                # if there are no points in the cluster, keep the old centroid
                cluster_centers.append(centroids[i])
            else:
                # calculate the new centroid as the mean of the points in the cluster and append it to the list
                cluster_centers.append(np.mean(points[indices], axis=0)[0])

        # if the difference between the old and new centroids is small enough
        # then we have converged
        if np.max(centroids - np.array(cluster_centers)) < 0.0001:
            break
        else:
            centroids = np.array(cluster_centers)

    return labels, centroids
```

Na začiatku funkcia vytvorí náhodne k centroidov, ktoré sú z daného priestoru. Následne ideme do cyklu ktorý bude iterovať do počtu zadaných iterácií. Vo vnútri cyklu najskôr vypočíta vzdialenosť všetkých bodov od všetkých centroidov. Následne priradíme bod ku najbližšiemu centroidu a uložíme si priradenie do poľa. Toto vykonáme pre všetky body. Ďalej vytvoríme pole kde je uložené či daný bod patrí danému clusteru (to znamená že zistíme body okolo daného centroidu). Následne iterujeme cez toto pole aktualizujeme pozíciu centroidu ak zmenil počet bodov v jeho clustery. Nakoniec overíme či sa nám vôbec robia zmeny (posúva sa centroid). Ak áno pokračujeme v hlavnom cykle. Ak nie tak vyjdeme z cyklu a vrátime pole centroidov a pole označení, ktoré body patria ktorému centroidu.

k_means_medoid(points, k, max_iter)

```
def k_means_medoid(points, k, max_iter):
    medoids = np.array([points[i] for i in range(k)])

    random_number = random.randint( a: 0, len(points))
    samples = np.array([points[(i * random_number) % len(points)] for i in range(20)])

    clusters, cost = get_cost(points, medoids)

    count = 0

    colors = np.array(np.random.randint(0, 255, size=(k, 4))) / 255
    colors[:, 3] = 1

    print("-----CLOSE THE PLOT TO CONTINUE-----")

    plt.title(f"Initial clusters")
    [plt.scatter(clusters[t][:, 0], clusters[t][:, 1], color=colors[t]) for t in range(k)]
    plt.scatter(medoids[:, 0], medoids[:, 1], facecolors='none', marker='o', s=100, edgecolors='black')
    plt.show()

    while True:
        swap = False
        print("Debug: iteration", count)
        for i in range(len(samples)):
            print("Debug: i", i)
            if not i in medoids:
                for j in range(k):
                    print("Debug: j", j)
                    tmp_meds = medoids.copy()
                    tmp_meds[j] = points[i]
                    clusters_, cost_ = get_cost(points, tmp_meds)

                    if cost_ < cost:
                        medoids = tmp_meds
                        cost = cost_
                        swap = True
                        clusters = clusters_
                        print(f"Medoids Changed to: {medoids}.")
        count += 1

        if count >= max_iter:
            print("End of the iterations.")
            break
        if not swap:
            print("No changes.")
            break

    return clusters, medoids
```

Na začiatku funkcia vytvorí náhodne k medoidov zo všetkých bodov, ktoré sú z daného priestoru. Následne si zoberieme 20 náhodných bodov, ktoré budú slúžiť na zmenu pozície medoidu. Zistíme si clustery pre medoidy a ich ceny. Spravíme vizualizáciu pre úvodné clustre. Ďalej zistíme clustery a ceny pre nové medoidy a porovnáme ich ceny. Ak sa nám cena zlepšila (znížila), tak nahradíme staré medoidy novými a pokračujeme v riadiacom cykle. Toto robíme do počtu maximálnych iterácií, ktoré sme zadali. Nakoniec nám funkcia vráti medoidy a body, ktoré im patria a tie sa v main funkcii vykreslia.

main() – Funkcia, ktorá sa zavolá na začiatku program a má v sebe nekonečný cyklus, ktorý slúži ako menu pre výber daných parametrov pri spustení určitého typu clusterovania. Následne podľa výberu zavolá potrebné funkcie a spraví vizualizáciu.

Testovanie a záver

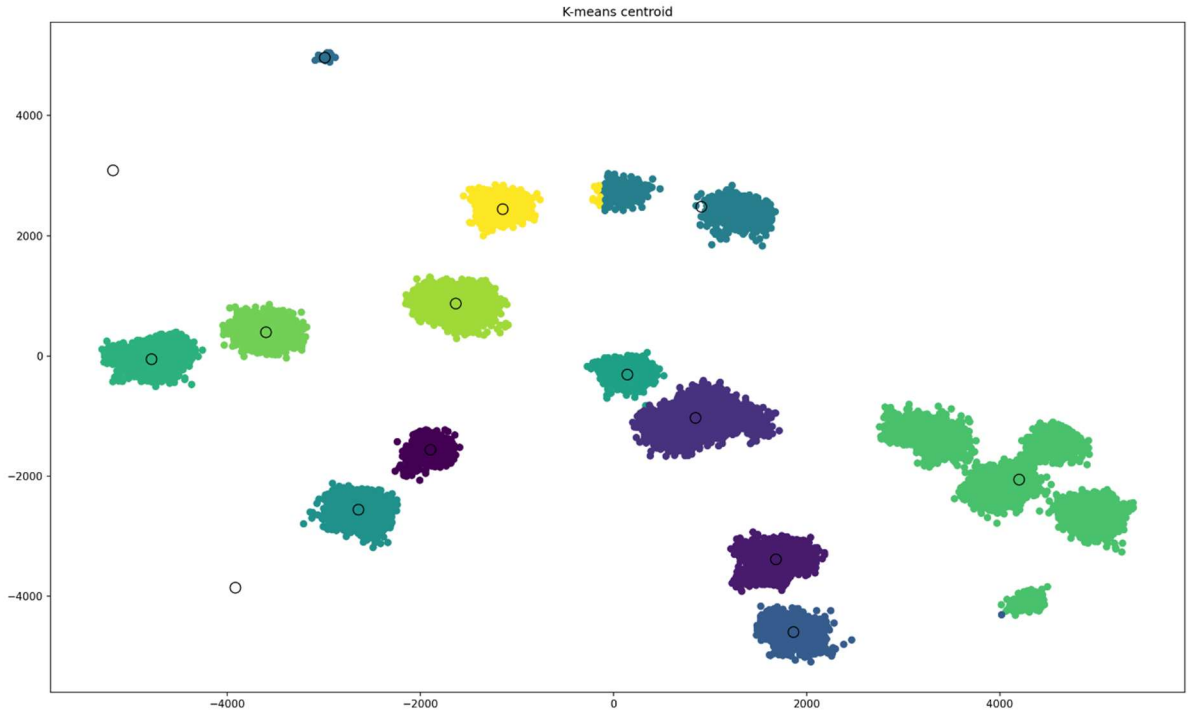
Implementáciu som testoval pre viacero rôznych parametrov. Ako príklad sem uvediem výsledky z oboch zhlukovačov pre tieto parametre:

K-means centroid:

- seed = 30
- k = 15
- max_iterations = 200

Výsledok:

- Unsuccesful



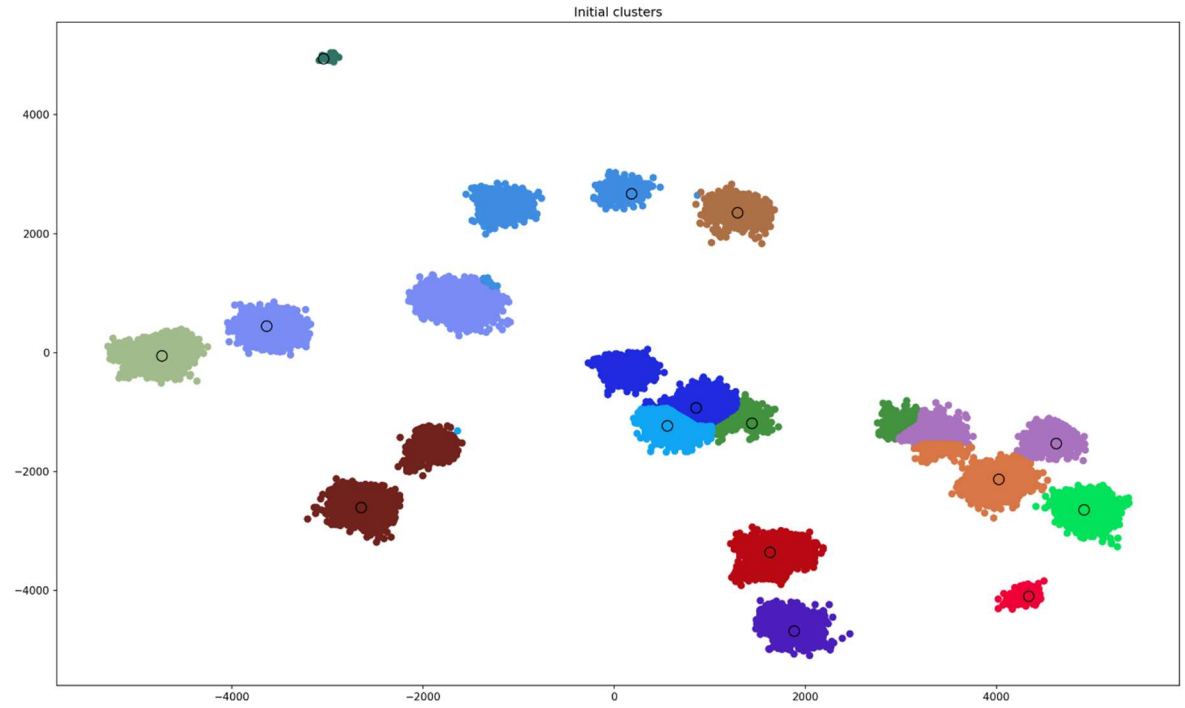
K-means medoid:

- seed = 30
- k = 15
- max_iterations = 2

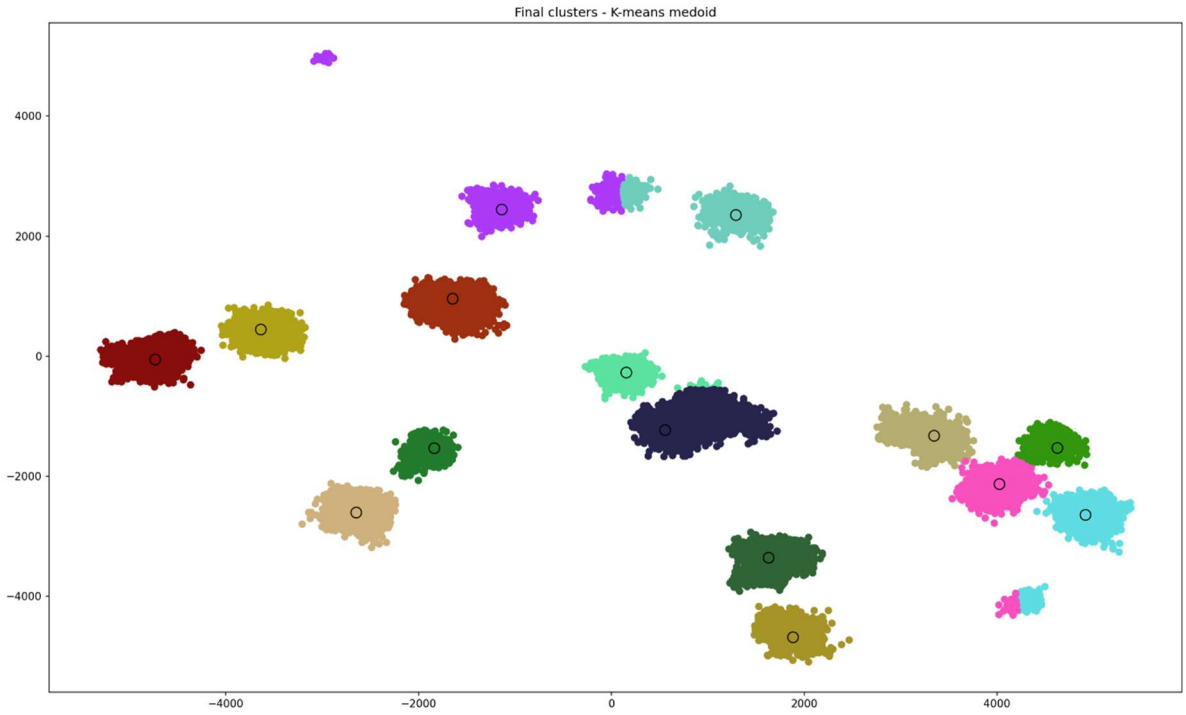
Výsledok:

- Succesful

Iniciálne clusterovanie (priradenie medoidov)



Koncové clustrovanie



K-means clusterovanie je výhodné z dôvodu jednoduchosti porozumieť mu a implementovať ho. Je relatívne rýchle a konverguje v danom počte krokov. Jeho nevýhodu je že nevie clusterovať skupiny alebo objekty, ktoré sú iného tvaru ako „kruhového“.

V mojej implementácii sa medoidom podarilo klasifikovať lepšie z dôvodu že medoidy sú fixné na nejaký bod v priestore, pričom centroidy sú iba nejakým miestom v strede bodov, ktoré mu patria.

Bohužiaľ sa mi nepodarilo implementovať aj tretí druh algoritmu. V kóde sú zakomentované časti, ktoré mu mali patriť avšak som nestihol ich spojazdniť.

Zdroje

http://www2.fiit.stuba.sk/~kapustik/Klastrovanie.html
https://www.youtube.com/watch?v=5w5iUbTlpMQ
https://www.youtube.com/watch?v=_aWzGGNrcic
https://www.youtube.com/watch?v=OFELCn-6r2o
https://www.youtube.com/watch?v=ChBxx4aR-bY
https://www.youtube.com/watch?v=MIWVfCcHzM4
https://dataqoil.com/2022/02/04/k-medoids-clustering-in-python-from-scratch/
https://www.geeksforgeeks.org/ml-k-medoids-clustering-with-example/
Prednášky z predmetu UI (Umelá inteligencia)